

Towards Formalising Trace Equivalence for Global and Local Types

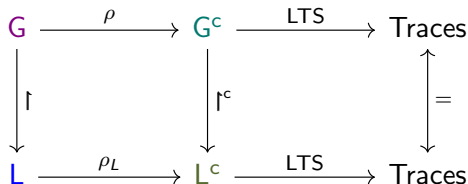
David Castro-Perez Francisco Ferreira Lorenzo Gheri
Nobuko Yoshida

Imperial College London

Verification of Session Types Workshop - 4th June 2020

Certifying the Semantics of Communication

This work is part of a bigger project for certifying, and reasoning about, programs in distributed systems.



- We formalize the meta-theory of multiparty session types¹.
- We use the Coq² Proof Assistant.

¹Deniérou P.-M., Yoshida N. (2013) Multiparty Compatibility in Communicating Automata: Characterisation and Synthesis of Global Session Types. ICALP 2013.

²<https://coq.inria.fr/>

Global and Local Types

Inductively defined by the following syntaxes:

G	$::=$	end	end type
	$ $	X	variable
	$ $	$\mu X.G$	recursion
	$ $	$p \rightarrow q : \{\ell_i(S_i).G_i\}_{i \in I}$	message
L	$::=$	end	end type
	$ $	X	variable
	$ $	$\mu X.L$	recursion
	$ $	$![q]; \{\ell_i(S_i).L_i\}_{i \in I}$	send type
	$ $	$?[p]; \{\ell_i(S_i).L_i\}_{i \in I}$	receive type

Types are assumed *closed* (no free variables) and recursion variables are always assumed *guarded* in types (namely types like $\mu X.X$ are not allowed).

Projection

Projection Rules:

- $\text{end} \upharpoonright r = \text{end};$ [PROJ-END]
- $X \upharpoonright r = X;$ [PROJ-VAR]
- $(\mu X.G) \upharpoonright r = \mu X.(G \upharpoonright r)$ [PROJ-REC]
- $r = p$ implies $p \rightarrow q : \{\ell_i(S_i).G_i\}_{i \in I} \upharpoonright r = ![q]; \{\ell_i(S_i).G_i \upharpoonright r\}_{i \in I};$ [PROJ-SEND]
- $r = q$ implies $p \rightarrow q : \{\ell_i(S_i).G_i\}_{i \in I} \upharpoonright r = ?[p]; \{\ell_i(S_i).G_i \upharpoonright r\}_{i \in I};$ [PROJ-RECV]
- $r \neq p, r \neq q$ and, for all $i, j \in I$, $G_i \upharpoonright r = G_j \upharpoonright r$; implies
 $p \rightarrow q : \{\ell_i(S_i).G_i\}_{i \in I} \upharpoonright r = G_1 \upharpoonright r;$ [PROJ-CONT]
- undefined otherwise.

Warm-Up!

A global type for a simple protocol:

$$G = p \rightarrow q : \ell(S). q \rightarrow r : \ell'(S'). \text{end}$$

and its projection on participant q:

$$G \restriction q = ?[p]; \ell(S). ![r]; \ell'(S'). \text{end}$$

Warm-Up!

A global type for a simple protocol:

$$G = p \rightarrow q : \ell(S).q \rightarrow r : \ell'(S').\text{end}$$

and its projection on participant q:

$$G \restriction q = ?[p]; \ell(S).![r]; \ell'(S').\text{end}$$

Do these global types have well defined projections?

$$p \rightarrow q : \{ \begin{array}{l} \ell_1(\text{oranges}).\text{end}, \\ \ell_2(\text{bananas}).q \rightarrow p : \ell_3(\text{pears}).\text{end} \end{array}$$

$$p \rightarrow q : \{ \begin{array}{l} \ell_1(\text{oranges}).\text{end}, \\ \ell_2(\text{bananas}).q \rightarrow r : \ell_3(\text{pears}).\text{end} \end{array}$$

$$p \rightarrow q : \{ \begin{array}{l} \ell_1(\text{oranges}).p \rightarrow r : \ell_3(\text{pears}).\text{end}, \\ \ell_2(\text{bananas}).q \rightarrow r : \ell_3(\text{pears}).\text{end} \end{array}$$

$$p \rightarrow q : \{ \begin{array}{l} \ell_1(\text{oranges}).q \rightarrow r : \ell_3(\text{pears}).\text{end}, \\ \ell_2(\text{bananas}).q \rightarrow r : \ell_3(\text{pears}).\text{end} \end{array}$$

Warm-Up!

A global type for a simple protocol:

$$G = p \rightarrow q : \ell(S).q \rightarrow r : \ell'(S').\text{end}$$

and its projection on participant q:

$$G \restriction q = ?[p]; \ell(S).![r]; \ell'(S').\text{end}$$

Do these global types have well defined projections?

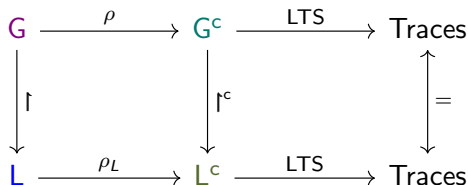
$$p \rightarrow q : \{ \begin{array}{l} \ell_1(\text{oranges}).\text{end}, \\ \ell_2(\text{bananas}).q \rightarrow p : \ell_3(\text{pears}).\text{end} \end{array} \} \quad \text{YES!}$$

$$p \rightarrow q : \{ \begin{array}{l} \ell_1(\text{oranges}).\text{end}, \\ \ell_2(\text{bananas}).q \rightarrow r : \ell_3(\text{pears}).\text{end} \end{array} \} \quad \text{NO!}$$

$$p \rightarrow q : \{ \begin{array}{l} \ell_1(\text{oranges}).p \rightarrow r : \ell_3(\text{pears}).\text{end}, \\ \ell_2(\text{bananas}).q \rightarrow r : \ell_3(\text{pears}).\text{end} \end{array} \} \quad \text{NO!}$$

$$p \rightarrow q : \{ \begin{array}{l} \ell_1(\text{oranges}).q \rightarrow r : \ell_3(\text{pears}).\text{end}, \\ \ell_2(\text{bananas}).q \rightarrow r : \ell_3(\text{pears}).\text{end} \end{array} \} \quad \text{YES!}$$

Certifying the Semantics of Communication



- Multiparty Session Types ✓
- Coinductive Trees (equi-recursive point of view)
- Semantics by Traces

An Equi-Recursive Viewpoint

"We adopt the *equi-recursive viewpoint*, i.e., we identify $\mu X.G$ and $G\{\mu X.G/X\}$."

An Equi-Recursive Viewpoint

“We adopt the *equi-recursive viewpoint*, i.e., we identify $\mu X.G$ and $G\{\mu X.G/X\}$.”

Example:

$\mu X.p \rightarrow q : \ell(S).X$ is “the same as” $p \rightarrow q : \ell(S).(\mu X.p \rightarrow q : \ell(S).X)$,

An Equi-Recursive Viewpoint

“We adopt the *equi-recursive viewpoint*, i.e., we identify $\mu X.G$ and $G\{\mu X.G/X\}$.”

Example:

$\mu X.p \rightarrow q : \ell(S).X$ is “the same as” $p \rightarrow q : \ell(S).(\mu X.p \rightarrow q : \ell(S).X)$,
which is “the same as” $p \rightarrow q : \ell(S).p \rightarrow q : \ell(S).(\mu X.p \rightarrow q : \ell(S).X)$,

An Equi-Recursive Viewpoint

“We adopt the *equi-recursive viewpoint*, i.e., we identify $\mu X.G$ and $G\{\mu X.G/X\}$.”

Example:

$\mu X.p \rightarrow q : \ell(S).X$ is “the same as” $p \rightarrow q : \ell(S).(\mu X.p \rightarrow q : \ell(S).X)$,
which is “the same as” $p \rightarrow q : \ell(S).p \rightarrow q : \ell(S).(\mu X.p \rightarrow q : \ell(S).X)$,
which is “the same as”
 $p \rightarrow q : \ell(S).p \rightarrow q : \ell(S).p \rightarrow q : \ell(S).(\mu X.p \rightarrow q : \ell(S).X)$,

An Equi-Recursive Viewpoint

"We adopt the *equi-recursive viewpoint*, i.e., we identify $\mu X.G$ and $G\{\mu X.G/X\}$."

Example:

$\mu X.p \rightarrow q : \ell(S).X$ is "the same as" $p \rightarrow q : \ell(S).(\mu X.p \rightarrow q : \ell(S).X)$,
which is "the same as" $p \rightarrow q : \ell(S).p \rightarrow q : \ell(S).(\mu X.p \rightarrow q : \ell(S).X)$,
which is "the same as"

$p \rightarrow q : \ell(S).p \rightarrow q : \ell(S).p \rightarrow q : \ell(S).(\mu X.p \rightarrow q : \ell(S).X)$,

which is "the same as"

$p \rightarrow q : \ell(S).p \rightarrow q : \ell(S).p \rightarrow q : \ell(S).p \rightarrow q : (\mu X.p \rightarrow q : \ell(S).X)$,

An Equi-Recursive Viewpoint

“We adopt the *equi-recursive viewpoint*, i.e., we identify $\mu X.G$ and $G\{\mu X.G/X\}$.”

Example:

$\mu X.p \rightarrow q : \ell(S).X$ is “the same as” $p \rightarrow q : \ell(S).(\mu X.p \rightarrow q : \ell(S).X)$,
which is “the same as” $p \rightarrow q : \ell(S).p \rightarrow q : \ell(S).(\mu X.p \rightarrow q : \ell(S).X)$,
which is “the same as”

$p \rightarrow q : \ell(S).p \rightarrow q : \ell(S).p \rightarrow q : \ell(S).(\mu X.p \rightarrow q : \ell(S).X)$,

which is “the same as”

$p \rightarrow q : \ell(S).p \rightarrow q : \ell(S).p \rightarrow q : \ell(S).p \rightarrow q : (\mu X.p \rightarrow q : \ell(S).X)$,

...

with a coinductive unrolling process!

Coinductive Asynchronous Trees

Global Trees:

$$\begin{array}{ll}
 G^c & ::= \text{end}^c & \text{end type} \\
 & | \quad p \rightarrow q : \{\ell_i(S_i).G^c_i\}_{i \in I} & \text{message send} \\
 & | \quad p \xrightarrow{\ell_i} q : \{\ell_i(S_i).G^c_i\}_{i \in I} & \text{message receive}
 \end{array}$$

Coinductive unrolling ρ :

$$\frac{}{\text{end } \rho \text{ end}^c} \quad \frac{G\{\mu X.G/X\} \rho G^c}{\mu X.G \rho G^c}$$

$$\frac{\forall i \in I. G_i \rho G^c_i}{p \rightarrow q : \{\ell_i(S_i).G_i\}_{i \in I} \rho p \rightarrow q : \{\ell_i(S_i).G^c_i\}_{i \in I}}$$

Coinductive Asynchronous Trees

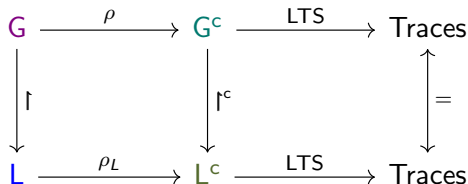
Global Trees:

$$\begin{array}{lll} G^c & ::= & \text{end}^c \quad \text{end type} \\ & | & p \rightarrow q : \{\ell_i(S_i).G^c_i\}_{i \in I} \quad \text{message send} \\ & | & p \xrightarrow{\ell_i} q : \{\ell_i(S_i).G^c_i\}_{i \in I} \quad \text{message receive} \end{array}$$

Local Trees:

$$\begin{array}{lll} L^c & ::= & \text{end}^c \quad \text{end type} \\ & | & !^c[p]; \{\ell_i(S_i).L^c_i\}_{i \in I} \quad \text{send type} \\ & | & ?^c[q]; \{\ell_i(S_i).L^c_i\}_{i \in I} \quad \text{receive type} \end{array}$$

Certifying the Semantics of Communication



- Multipart Session Types ✓
- Coinductive Trees (equi-recursive point of view) ✓
- Semantics by Traces

Asynchronous Semantics

p sends a message to q with label ℓ :

$$p \rightarrow q : \ell(S).G^c$$

We keep track of this communication with a queue:

ϵ

Asynchronous Semantics

p sends a message to q with label ℓ :

$$p \rightarrow q : \ell(S).G^c \xrightarrow{\text{step } \ell} p \xrightarrow{\ell} q : \ell(S).G^c$$

We keep track of this communication with a queue:

$$\epsilon \xrightarrow{\text{enqueue}} [(\ell, S)]$$

Asynchronous Semantics

p sends a message to q with label ℓ :

$$p \rightarrow q : \ell(S).G^c \xrightarrow{\text{step } \ell} p \xrightarrow{\ell} q : \ell(S).G^c \xrightarrow{\text{step } \ell} G^c$$

We keep track of this communication with a queue:

$$\epsilon \xrightarrow{\text{enqueue}} [(\ell, S)] \xrightarrow{\text{dequeue}} \epsilon$$

Asynchronous Semantics

p sends a message to q with label ℓ :

$$p \rightarrow q : \ell(S).G^c \xrightarrow{\text{step } \ell} p \xrightarrow{\ell} q : \ell(S).G^c \xrightarrow{\text{step } \ell} G^c$$

We keep track of this communication with a queue:

$$\epsilon \xrightarrow{\text{enqueue}} [(\ell, S)] \xrightarrow{\text{dequeue}} \epsilon$$

We have such a queue for each pair of participants.

Queues for Asynchronous Local Coordination

p sends and q receive:

$$!^c[q];\ell(S).L^c \xrightarrow{\text{step}} L^c$$

$$Q(p, q) = \epsilon \xrightarrow{\text{enqueue}} Q(p, q) = [(\ell, S)] \xrightarrow{\text{dequeue}} Q(p, q) = \epsilon$$

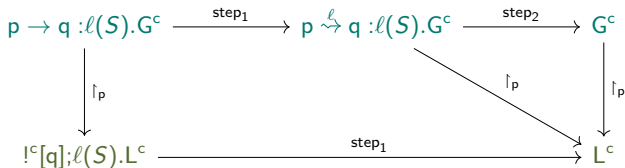
$$?^c[p];\ell(S).L^{c'} \xrightarrow{\text{step}} L^{c'}$$

Queue Environments

A *queue environment* is a finitely supported function Q of type $\text{role} \times \text{role} \rightarrow \mathcal{W}$, where \mathcal{W} is the set of finite words w (*queues*) on the alphabet $\text{labels} \times \text{sorts}$.

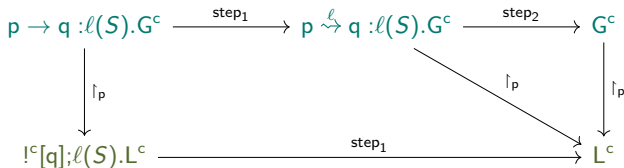
Global and Local Steps

p sends:

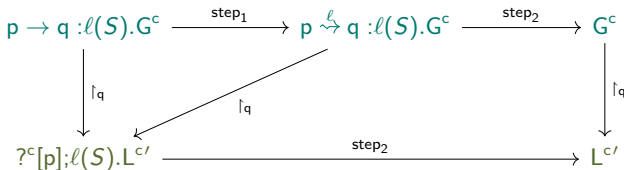


Global and Local Steps

p sends:



q receives:



“Local” Environments

We want to consider altogether the different local types involved in the communication.

Environments for Local Types

An *environment for local types* is a finitely supported function E of type $\text{role} \rightarrow \text{l_ty}^c$.

“Local” Environments

We want to consider altogether the different local types involved in the communication.

Environments for Local Types

An *environment for local types* is a finitely supported function E of type $\text{role} \rightarrow \text{l_ty}^c$.

Why?!

“Local” Environments

We want to consider altogether the different local types involved in the communication.

Environments for Local Types

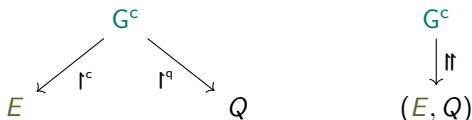
An *environment for local types* is a finitely supported function E of type $\text{role} \rightarrow \text{lt}_p^c$.

Why?!

It will be $E(p) = L_p^c$, where $G^c \vdash_p^c L_p^c$.

One-Shot Projection

We get the one-shot projection of a global type both on environments of local types and on queue environments.



Step Results

Theorem (Step Soundness)

If $G^c \Vdash (E, Q)$ and $G^c \xrightarrow{\text{step}} G^{c'}$, then there exist E' and Q' , such that $G^{c'} \Vdash (E', Q')$ and $(E, Q) \xrightarrow{\text{step}} (E', Q')$.

Theorem (Step Completeness)

If $G^c \Vdash (E, Q)$ and $(E, Q) \xrightarrow{\text{step}} (E', Q')$, then there exists $G^{c'}$, such that $G^{c'} \Vdash (E', Q')$ and $G^c \xrightarrow{\text{step}} G^{c'}$.

Labelled Transition System

Keeping track of messages...

Actions

An *action* a is an object of the shape:

- either $pq!(\ell, S)$ (send action),
- or $pq?(\ell, S)$ (receive action).

Traces

A *trace* t is a coinductive, possibly infinite, stream of actions:

- $a_1 a_2 a_3 \dots$ is a trace.

Traces for Trees

p sends and q receives:

$$p \rightarrow q : \ell(S).G^c \xrightarrow{\text{step}_1} p \xrightarrow{\ell} q : \ell(S).G^c \xrightarrow{\text{step}_2} G^c$$

Let t be a trace for G^c ,

$$pq!(\ell, S)pq?(\ell, S)t \xleftarrow{\text{step}_1} pq?(\ell, S)t \xleftarrow{\text{step}_2} t$$

Non-Determinism

Let us consider two different executions:

- $p \rightarrow q : \{\ell_i(S_i).G^c_i\}_{i \in I} \xrightarrow{\text{step } \ell_1} p \xrightarrow{\ell_1} q : \{\ell_i(S_i).G^c_i\}_{i \in I} \xrightarrow{\text{step } \ell_1} G^c_1$
- $p \rightarrow q : \{\ell_i(S_i).G^c_i\}_{i \in I} \xrightarrow{\text{step } \ell_2} p \xrightarrow{\ell_2} q : \{\ell_i(S_i).G^c_i\}_{i \in I} \xrightarrow{\text{step } \ell_2} G^c_2$

If t_1 is a trace admissible for G^c_1 and t_2 is admissible for G^c_2 , both

- $pq!(\ell_1, S_1)pq?(\ell_1, S_1)t_1$ and
- $pq!(\ell_2, S_2)pq?(\ell_2, S_2)t_2$

are admissible for $p \rightarrow q : \{\ell_i(S_i).G^c_i\}_{i \in I}$.

Trace Equivalence

And finally...

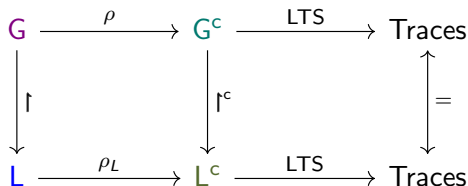
Trace Equivalence

And finally...

Theorem (Trace Equivalence)

If $G^c \vdash^c E$ then the set of traces admissible for G^c is equal to the set of traces admissible for E .

Certifying the Semantics of Communication



- Multiparty Session Types ✓
- Coinductive Trees (equi-recursive point of view) ✓
- Semantics by Traces ✓ (almost)

Things We Used

Formalisation in the Coq³ Proof Assistant, in particular we have used:

- the SSReflect⁴ proof language;
- the Mathematical Components⁵ libraries;
- the PaCo library for parametrized coinduction⁶.

³<https://coq.inria.fr/>

⁴<https://coq.inria.fr/refman/proof-engine/ssreflect-proof-language.html>

⁵<https://math-comp.github.io/>

⁶<https://github.com/snu-sf/paco>

Conclusion

Things we have got:

- a formalisation of the metatheory of mutliparty session types in Coq
- two birds with a (coinductive) stone: equi-recursion and no bindings
- (non-deterministic) semantics through labelled transition systems
- types that are ready for typing!

From here what?

- a more comprehensive version of MPST (e.g., with merge operator)
- communicating finite state automata
- ... the future is unwritten!

Conclusion

Things we have got:

- a formalisation of the metatheory of mutliparty session types in Coq
- two birds with a (coinductive) stone: equi-recursion and no bindings
- (non-deterministic) semantics through labelled transition systems
- types that are ready for typing! **Please attend Francisco's talk! :)**

From here what?

- a more comprehensive version of MPST (e.g., with merge operator)
- communicating finite state automata
- ... the future is unwritten!

Thank You!